Splotch: High Performance Visualization using MPI, OpenMP and CUDA

Klaus Dolag (Munich University Observatory)
Martin Reinecke (MPA, Garching)
Claudio Gheller (CSCS, Switzerland),
Marzia Rivi (CINECA, Italy), Mel Krokos (University of Portsmouth),

Motivations

Numerical simulations in astrophysics and fluid-dynamics produce huge (tens or hundreds of Terabytes) and complex data.

Effective tools to explore and analyze these data are required.

Visualization is the most immediate and intuitive way of inspecting large-scale data sets:

- o accelerate and simplifying cognitive process
- o focus on subsamples and features of interest
- o detect correlations and special characteristics of data
- o ....

Visualization is also an effective instrument for education and outreach.

Objectives

Our main focus is on **huge datasets** (tens to hundreds of terabytes) that cannot be visualized interactively and/or using “standard” visualization facilities.

We want to use **HPC resources**, exploiting hybrid architectures that allow to load and process data in an acceptable time.

Sort of “**brute force approach**”, where all other solutions fail (or cannot be easily used).

We have focused on the following paradigms:

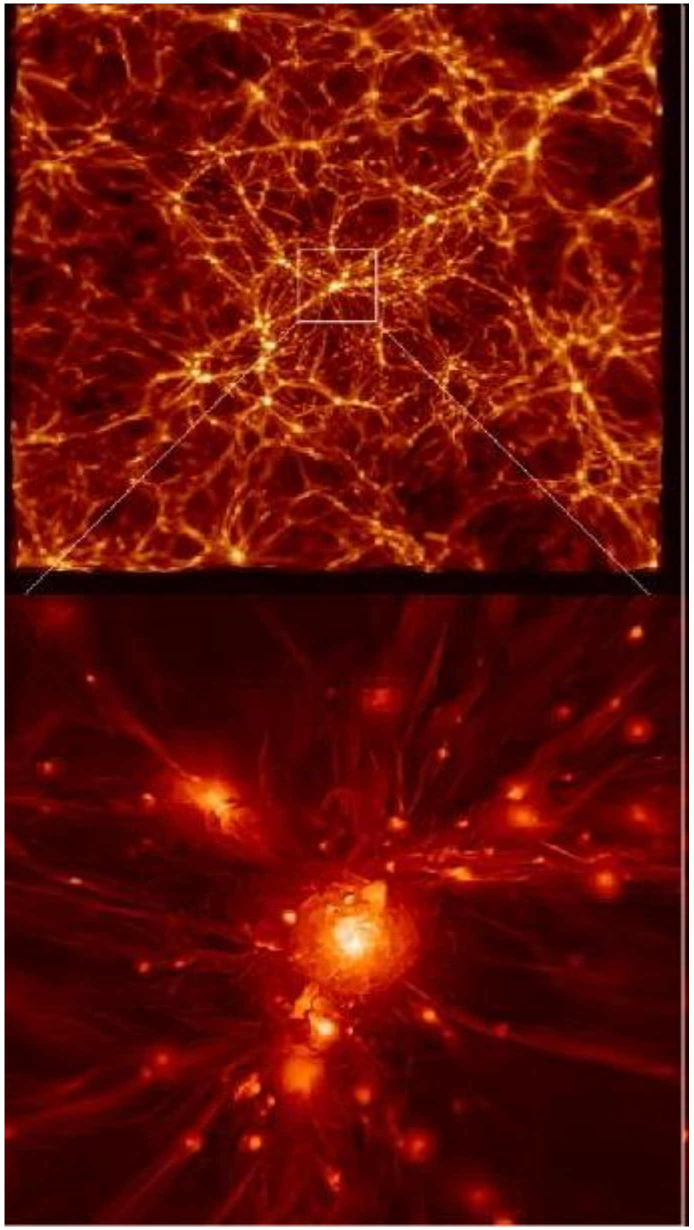
- MPI (internode)
- OpenMP (intranode)
- (CUDA/OpenCL for GPUs)

Splotch: overview

Splotch is a **ray-tracing algorithm** for effective visualization of large-scale astrophysical datasets coming from particle-based simulations. Based on (approximate) solution of the **radiative transport equation**

$$\frac{d\mathbf{I}(\mathbf{x})}{dx} = (\mathbf{E_p} - \mathbf{A_p}\mathbf{I}(\mathbf{x}))\rho_\mathbf{p}(\mathbf{x})$$

where the density is calculated smoothing the particle quantity on a “proper” neighborhood, by a Gaussian like distribution function.



Splotch: overview (cont.ed)

Basic Splotch steps:

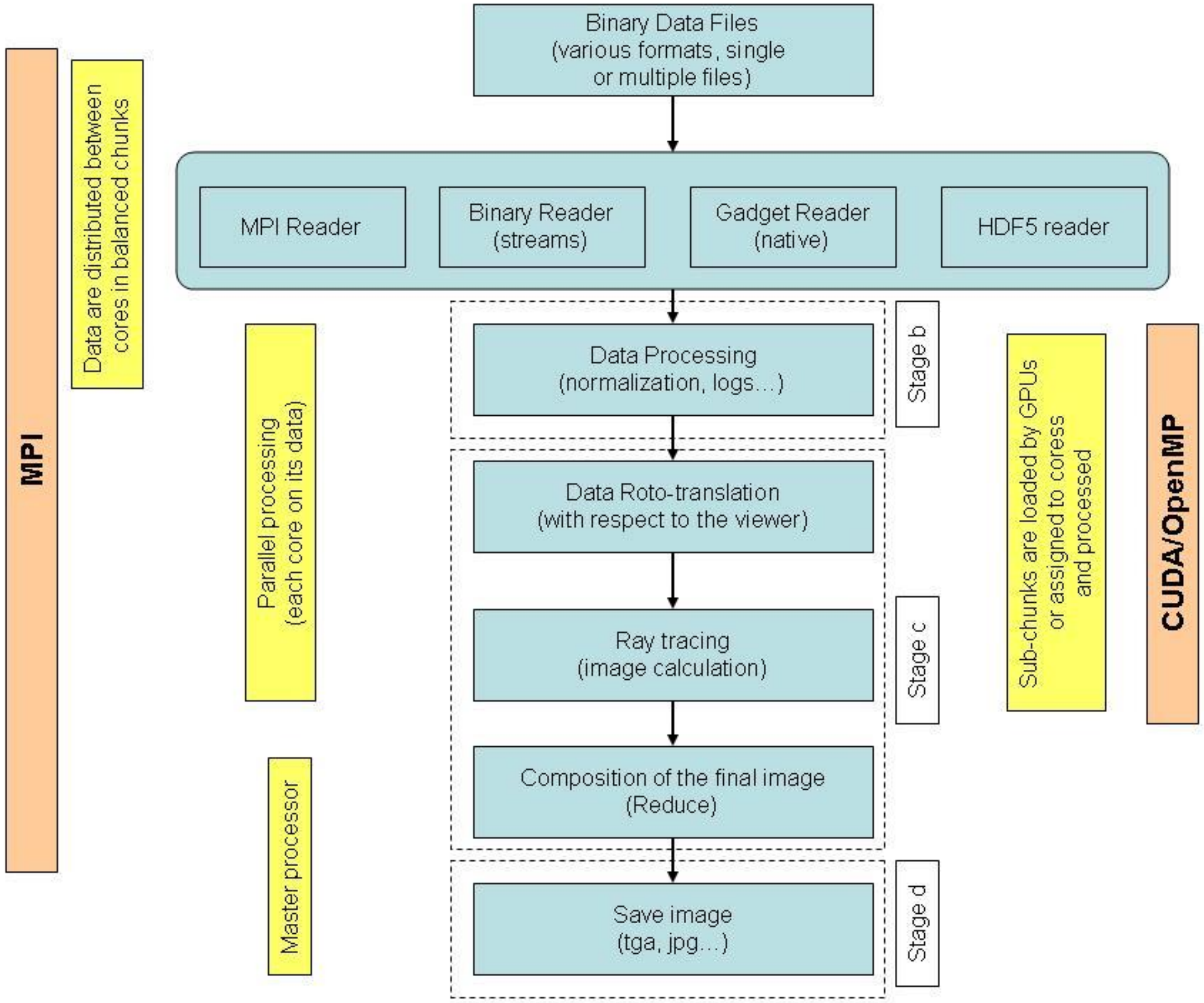
1. **Read** data
2. **Process** (normalization, logs…) data
3. **Set the point of view** (roto-translation) and **prospective projection** in the xy-plane
4. **Color** (associate a emission and absorption to each particle)
5. **Render** (solve radiative transfert equation producing partial image)
6. **Save final image** (tga, jpeg)

C++ procedural (not OO) code, completely self contained (no need for external libraries – except those for parallelization)

Specific care spent to high performance and memory consumption.

No configure procedure available… Set Makefile by hands…

Workflow of the Splotch algorithm



MPI
Data are distributed between cores in balanced chunks
Binary Data Files (various formats, single or multiple files)
MPI Reader
Binary Reader (streams)
Gadget Reader (native)
HDF5 reader
Parallel processing (each core on its data)
Data Processing (normalization, logs…)
Stage b
Data Roto-translation (with respect to the viewer)
Ray tracing (image calculation)
Stage c
Composition of the final image (Reduce)
Master processor
Save image (tga, jpg…)
Stage d
Sub-chunks are loaded by GPUs or assigned to coress and processed
CUDA/OpenMP

Splotch: readers

Data sets contain properties of the simulation particles, such as position, velocity, mass density, smoothing length, etc.

Formats supported are:

- **Tabular binary** (each row contains all properties of a single particle)
- **Blocks binary** (each block contains a single property for all particles)
- **Gadget** (format 1, format 2, hdf5) output format
- **Mesh** (each particle corresponds to a cell of a 3-dimensional grid)
- **HDF5**

Blocks format allows to enable MPI I/O reader where each process has a different view of the file, so that simultaneous collective writing/reading of non-contiguous interleaved data are allowed

Splotch: MPI parallelization

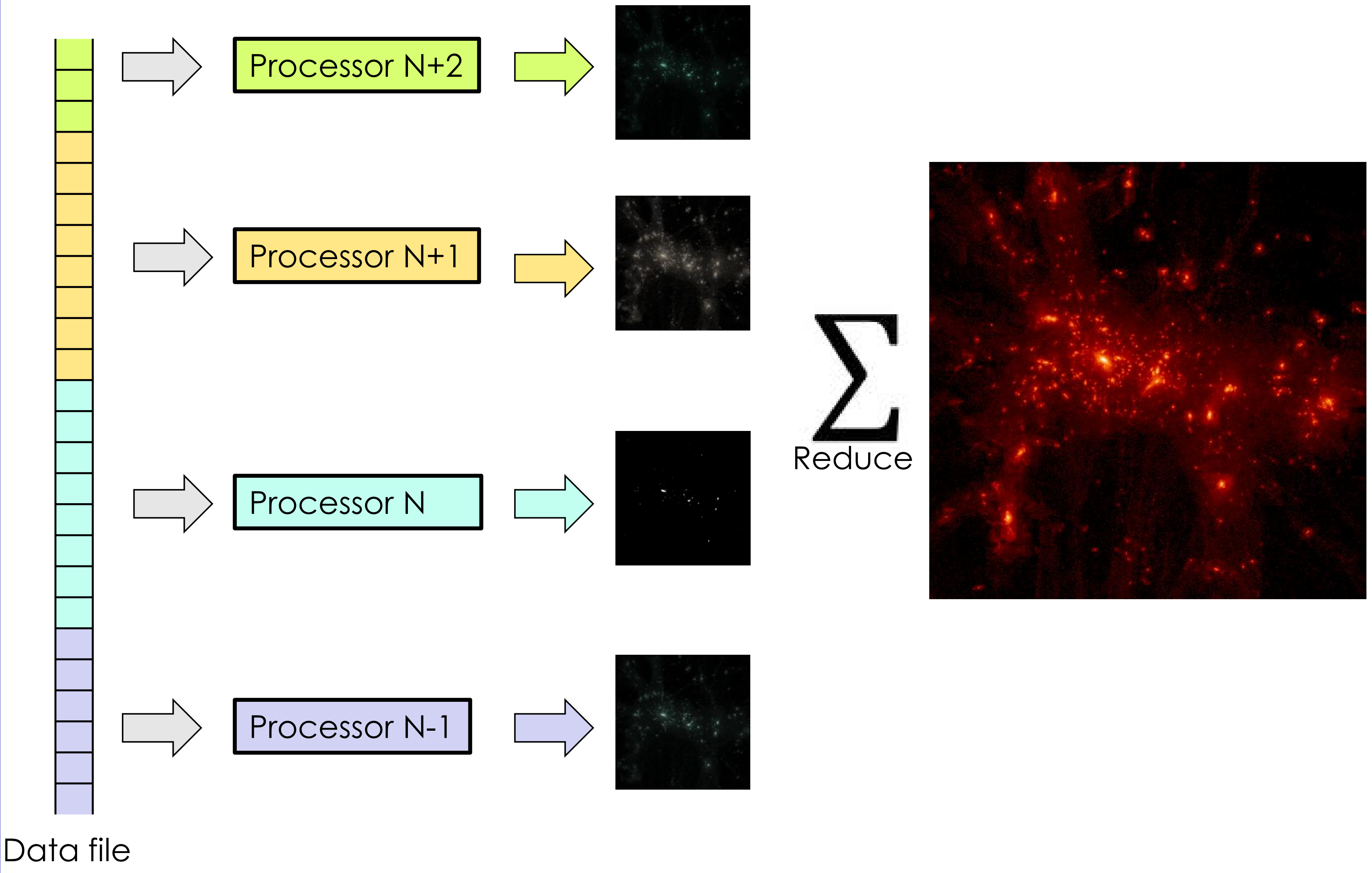
Assuming $E_p = A_p$ in the transport equation $\frac{d\mathbf{I}(\mathbf{x})}{dx} = (\mathbf{E_p} - \mathbf{A_p}\mathbf{I}(\mathbf{x}))\rho_\mathbf{p}(\mathbf{x})$
the algorithm can be easily parallelized using a SIMD approach based on MPI. Critical steps are data load and interpolation (all other steps are embarrassingly parallel)

Two main data structures:
particles (the input dataset) – can be huge
Image – small compared to particle set but problematic for GPU

Particles are distributed between processors at reading time.
Different implementations to support several file formats and to tune the performances (MPI-I/O, parallel streams, read-communicator…)

Image is replicated on all processors (typical 1000x1000 image is 3 MB)
Final image reduced between all processors

MPI parallel rendering process



MPI benchmarks on IBM-Power6

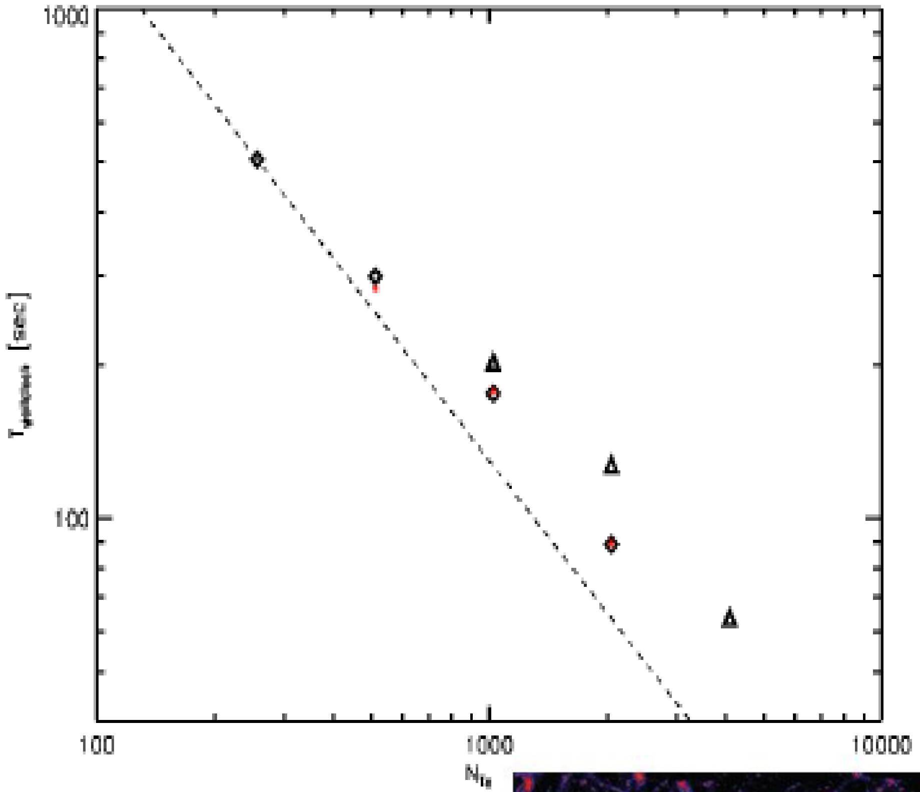
Hardware architecture UNIX-AIX SP6 system:

- Cluster of 168 Power6 575 computing nodes
- 32 cores and 128 GBytes per node
- 2 possible schedule modes for execution:
  - ST (Single Thread)
  - SMT (Simultaneous Multi-Threading)

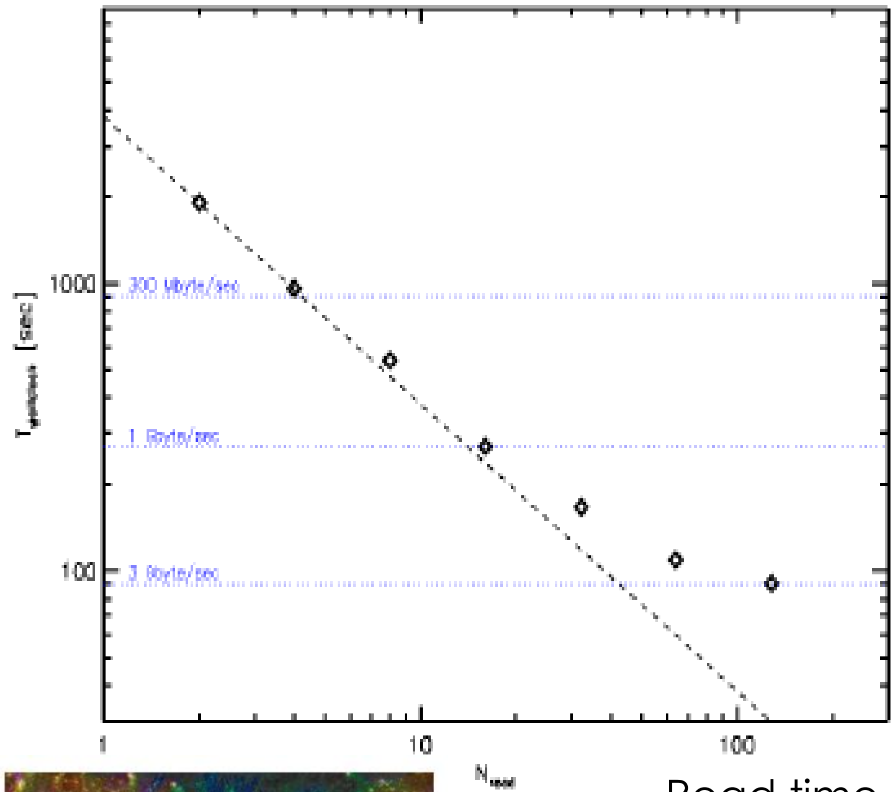
Benchmark data set:

- Millennium II simulation output containing about 10 billion particles

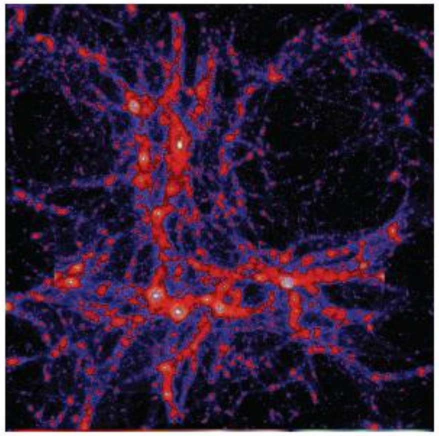
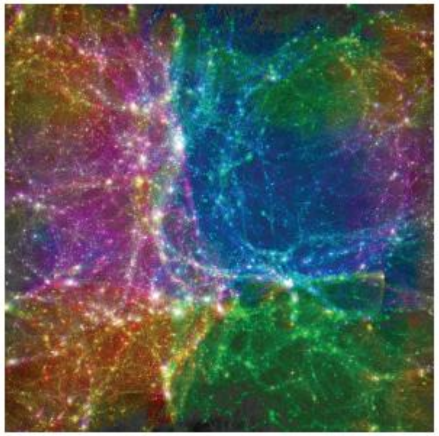
Splotch: benchmark results Millennium run



Total wallclock time minus read time



Read time

Millenium II simulation

diamonds = ST mode

traingles = SMT mode

OpenMP

Simple parallelization, consisting in in the distribution of work of the main loops among different threads.

This is fine in all the “preliminary” steps…

…but not for the rendering part, where images are created.

Images can be shared or private:

Shared → low memory request but race conditions

Private → wasting memory and serialized process (critical regions)

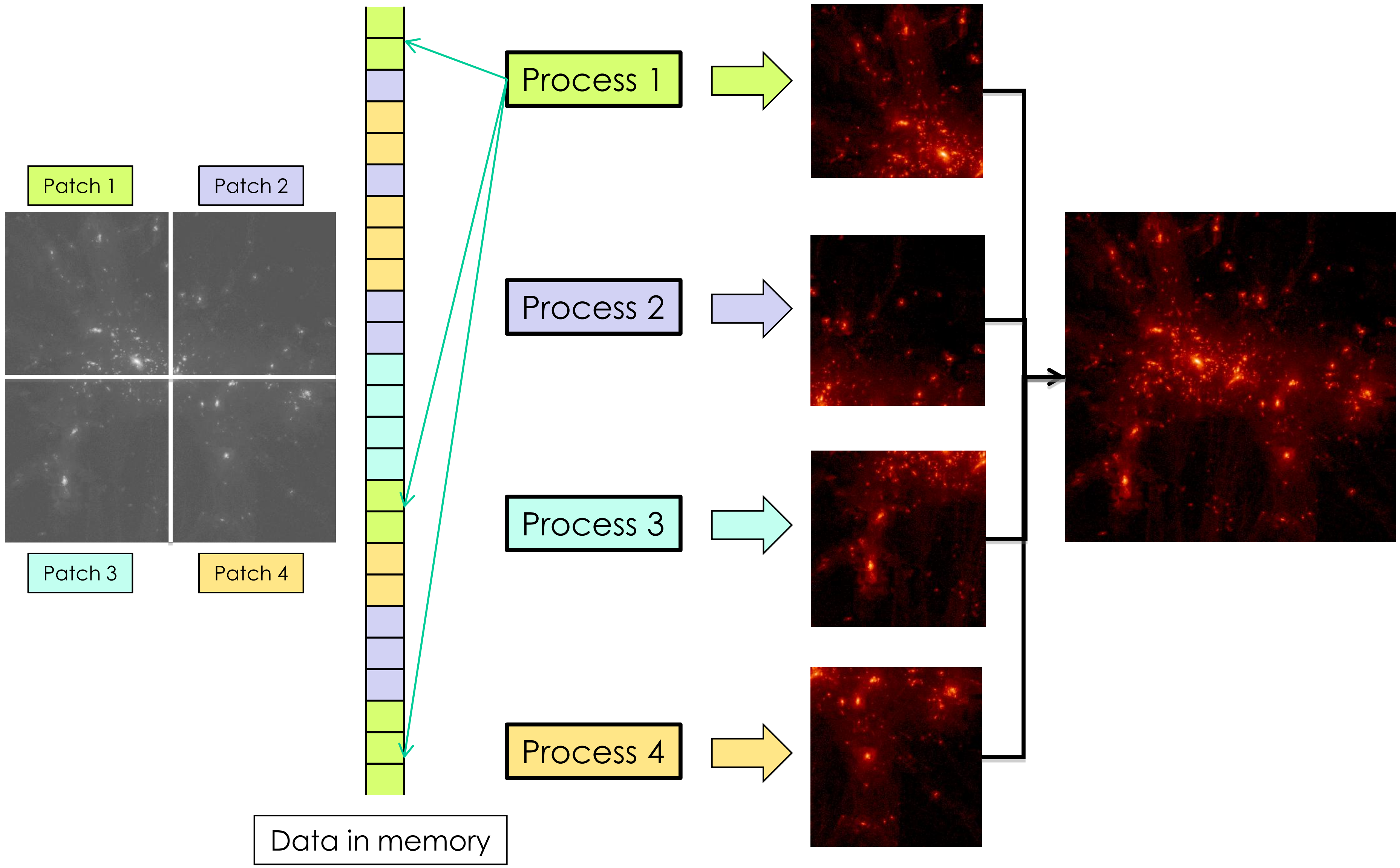
Adopted solution: split the image between threads, in chunks

1. establish which particles contribute to each chunk of the image
2. each thread renders its own portion of the image

Small chunks (number of chunks >> number of threads) lead to better load balancing.

- **No race conditions**
- **Can be used with MPI**

OpenMP parallel rendering process



OpenMP performances

number of OpenMP threads	1 (serial)	2	4	8
Range	6.66	3.40	2.00	1.53
Transform	5.77	2.91	1.49	0.76
Color	5.24	2.64	1.35	0.88
Render	19.20	11.44	7.66	5.31
Tot. Compute	36.87	19.39	12.50	7.48

Scaling of the CPU time (total wallclock time) with the number of OpenMP threads used for visualizing 100 million particles for an 800x800 pixel display

Splotch: CUDA implementation & issues

The **CUDA approach** consists in:

- Have a copy of the image on each GPU
- Push chunks of particles on GPU memory
- Create a thread for each particle

Issues (for implementation and performance):

1. each particle influences different number of screen pixels, therefore unbalanced granularity can compromise execution times during rendering phase;
2. each particle access "randomly" the image, therefore possible concurrent access to the same pixel may occur.

Solutions:

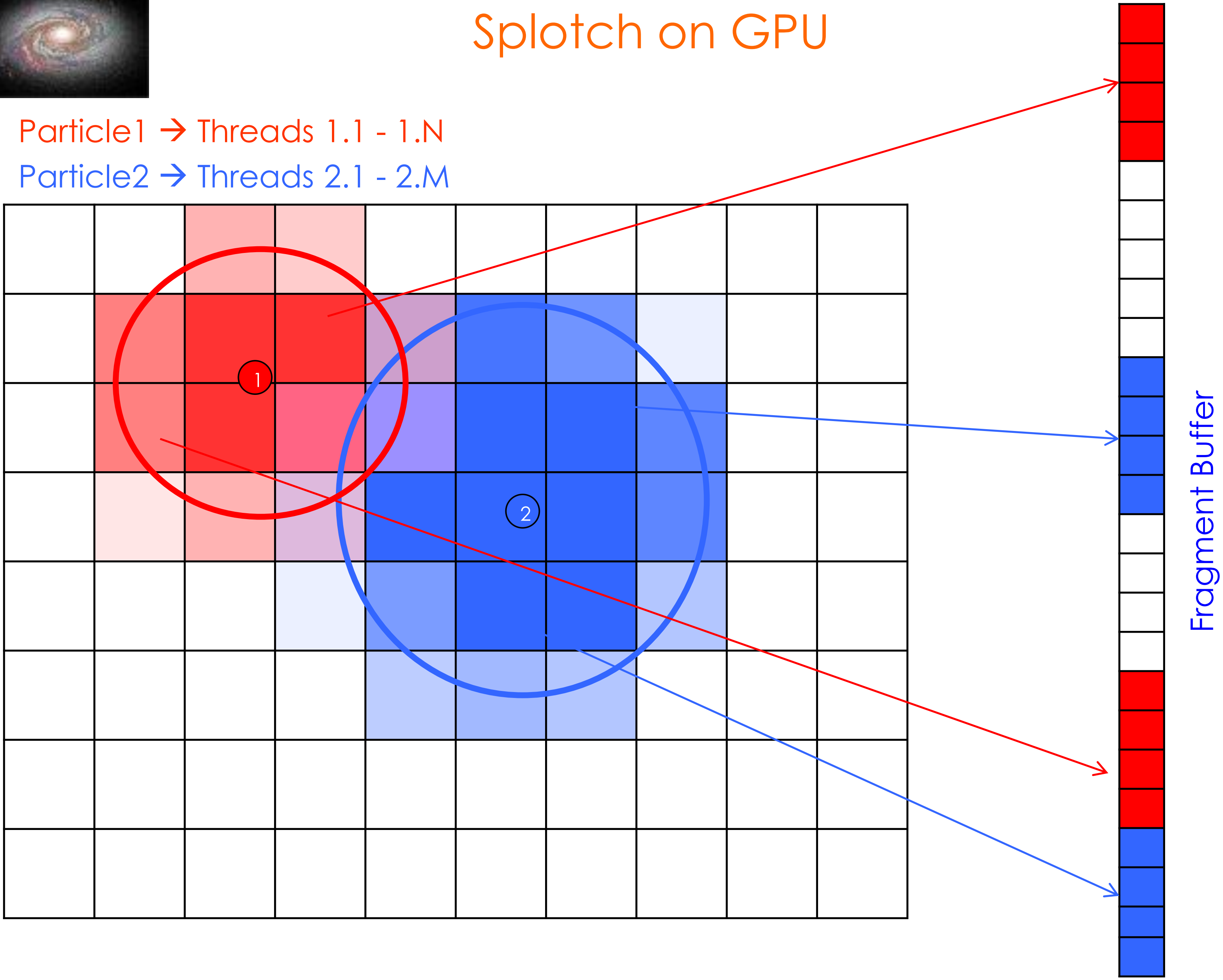
1. Split "big" particles in "smaller" influencing regions of the same size.
2. have a copy of the image on each host and use a fragment buffer on the GPU to store the pixels associated to each particle;

Remaining major problem: memory management !!!

Splotch on GPU

Particle1 → Threads 1.1 - 1.N

Particle2 → Threads 2.1 - 2.M



Splotch: benchmark MPI+CUDA on Linux cluster

Hardware architecture IBM PLX Linux cluster:

- 274 compute nodes, each one contains 2 Nehalem quad-core Intel Xeon X5550 processors, with a clock of 2.66GHz, and have 24GB of memory (3GB per core).
- 12 fat nodes containing 2 Nehalem quad-core Intel Xeon X5570 processors, with a clock of 2.66GHz and 128GB of memory.
- 4 out of the 12 fat nodes are equipped with a graphic card NVIDIA Quadro Plex 2200 S4 (GPUs with 4Gb of memory).
- internal Network: Infiniband with 4 QDR switches

Benchmark data set: previous 100M data set.

Splotch: benchmark CUDA on Linux cluster

Optimization: we can exploit CPU, during its latence time, to render a subset of particles. Cuda version allows to launch parallel threads rendering subsets of particles, one executed by the CPU host (if enabled) and the others by different GPUs.
The percentage of particles distributed between host and devices is important to get the best performance.

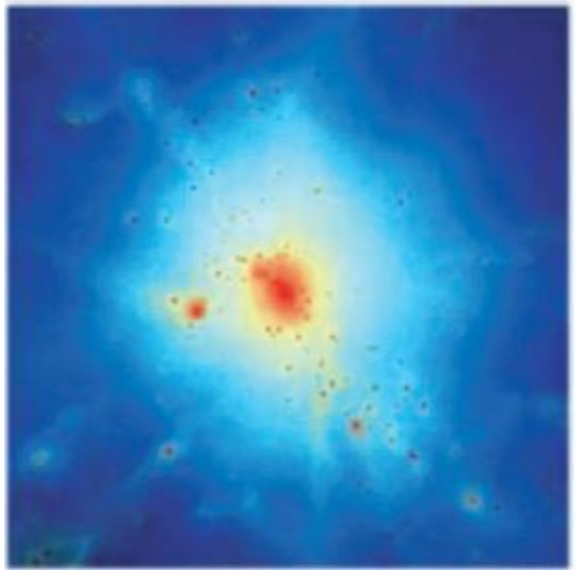
	Particles distribution between CPU and GPU (Time in sec.)					
	Test A		Test B		Test C	
	cpu 30%	gpu 70%	cpu 40%	gpu 60%	cpu 50%	gpu 50%
Range	2.05	1.45	2.69	1.24	3.39	1.03
Transform	1.28	0.73	1.70	0.62	2.13	0.52
Color	1.68	2.46	2.25	2.12	2.80	1.76
Render	5.10	4.54	6.80	3.89	8.56	3.26
Copy H-D		7.20		5.97		5.02
Tot. Compute	10.11	17.57	13.44	14.88	16.88	12.45

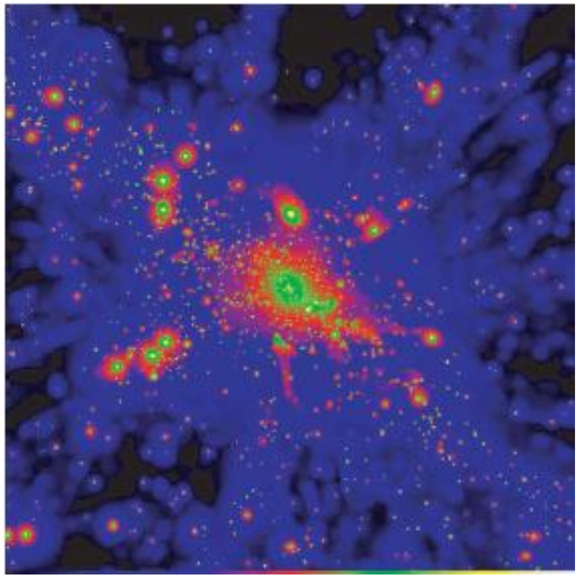
Splotch: benchmark MPI+CUDA on Linux cluster

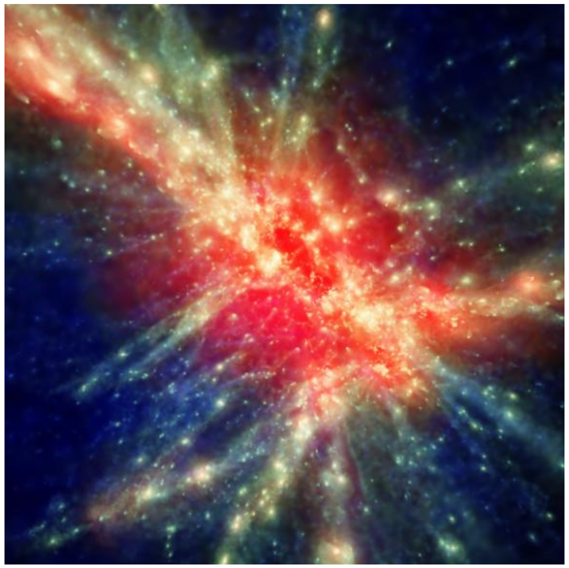
MPI process type	CPU time (sec.)		GPU time (sec.)		GPU+CPU	
Num of processes	1	2	1	2	1	2
Range	6.89	3.89	2.07	1.04	2.69	1.47
Transform	4.60	2.35	1.04	0.52	1.70	0.87
Color	5.45	2.80	3.52	1.79	2.25	1.13
Render	18.42	9.33	6.51	3.20	6.80	3.40
Copy H-D			10.24	5.85	5.97	3.12
Total Compute	35.36	17.7	25.08	13.64	14.88	7.57

- Linear scalability for all configurations.
- CUDA offers about 29% performance gain and host+device configuartion offers about 58% performance gain with respect to CPU.
- Considerably time spent for host-device copy operations. Number of copies depend on the **size of the dataset** and the **size of the GPU memory.**
- CUDA offers maximum gains when there is a large rendering calculation involved and low data copy operation between host and device.

Image Modes





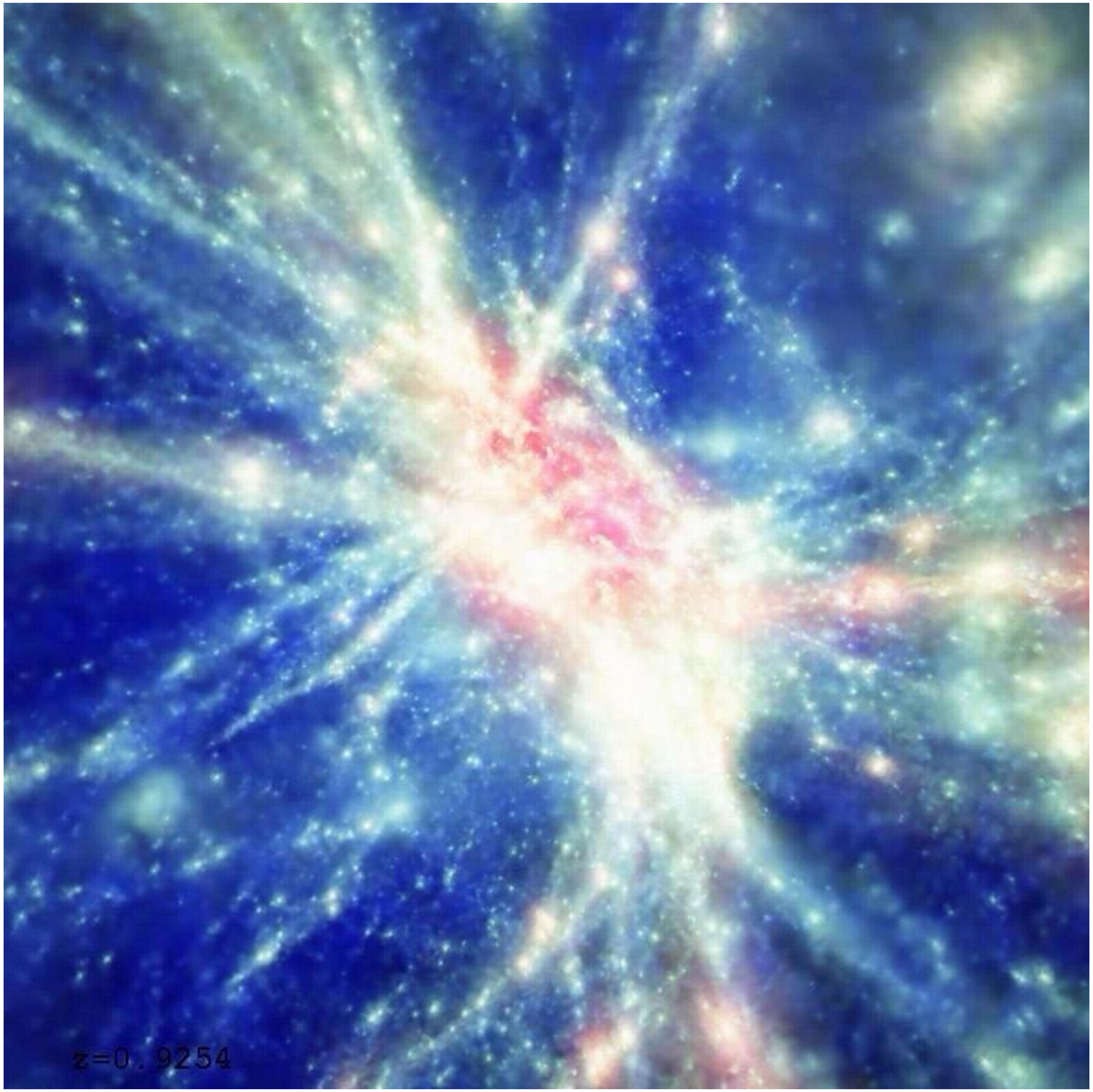


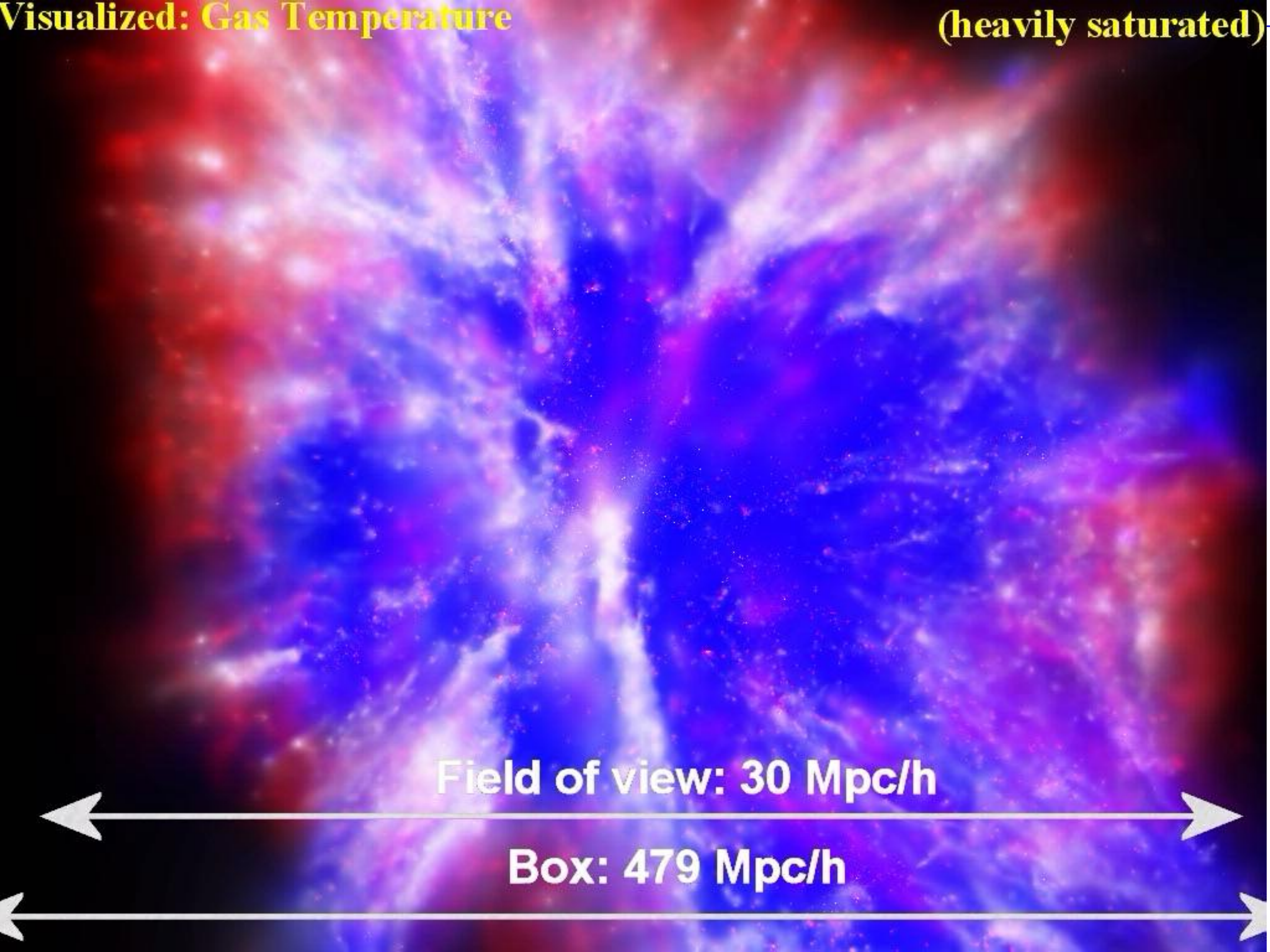
Sorting particles by:

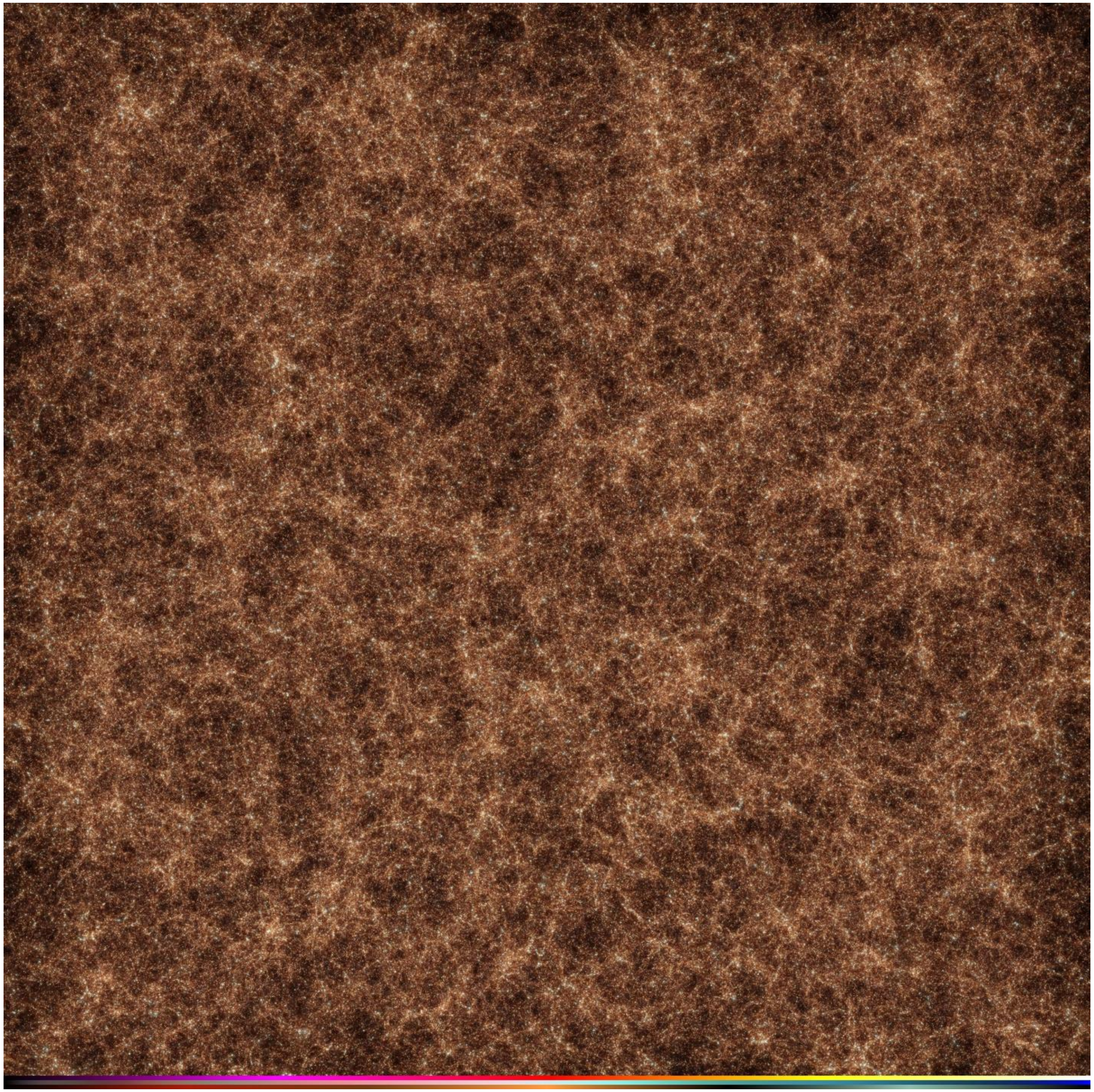
- Color (radiative transfer equation)
- Value (emphasize structure)

Coloring particles by:

- Scalar value (lookup table)
- Vector (direct mapping to RGB)

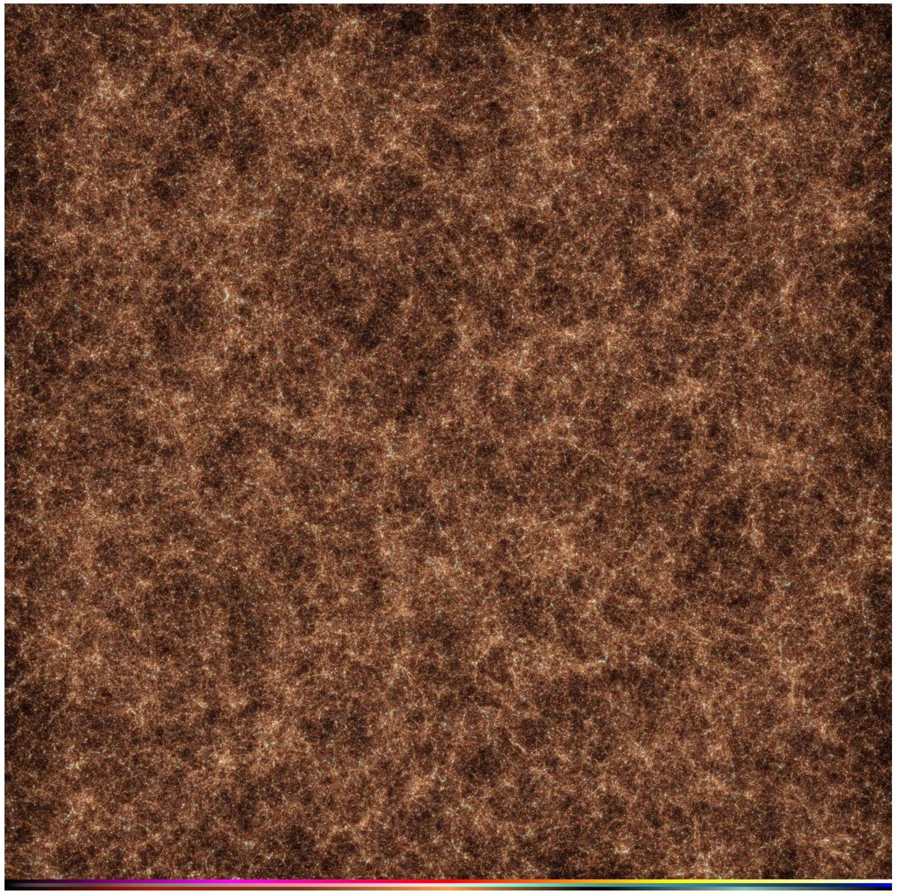

z=0.9254

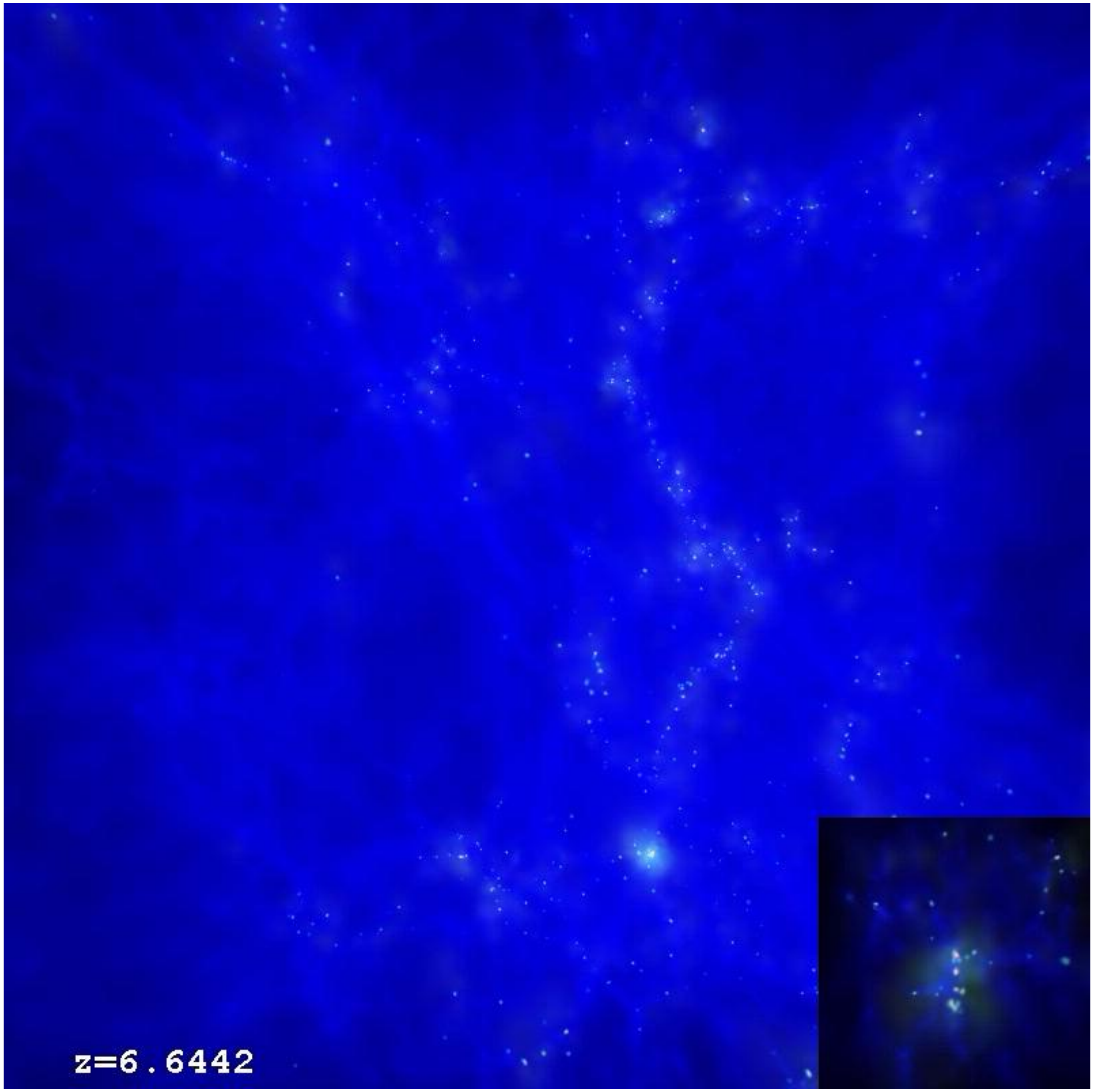

Visualized: Gas Temperature
(heavily saturated)
Field of view: 30 Mpc/h
Box: 479 Mpc/h



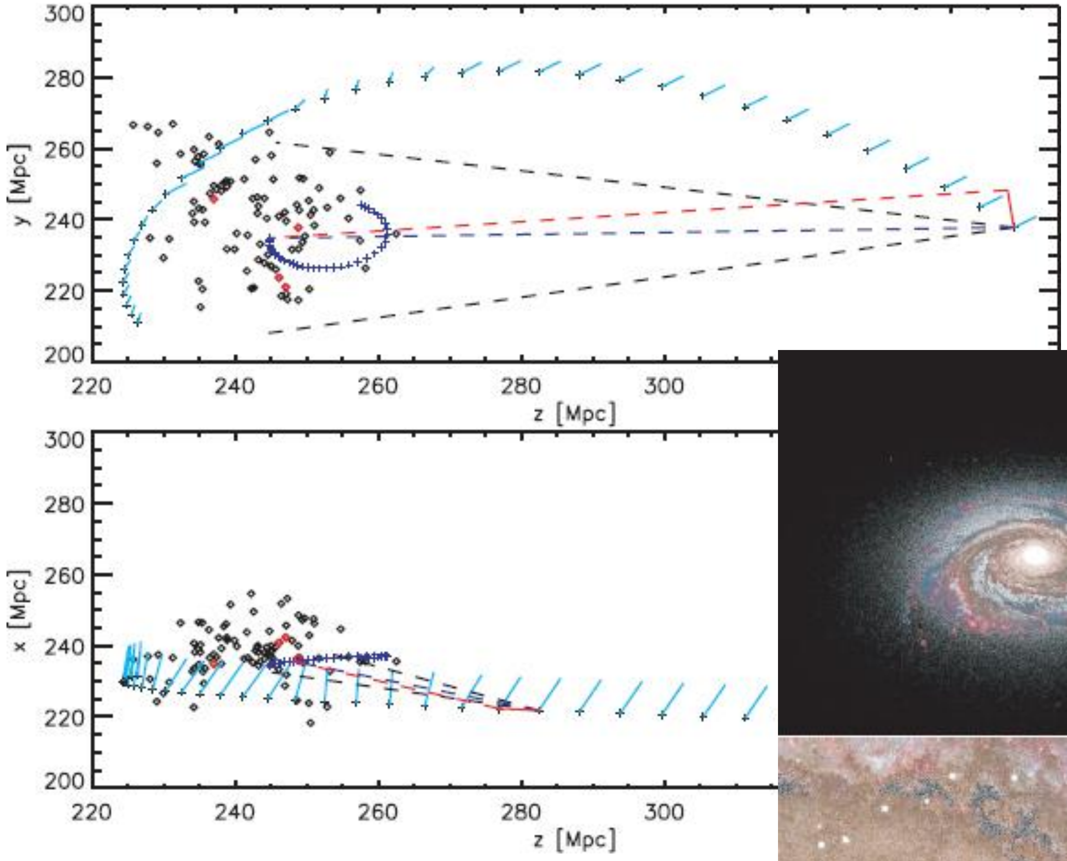
Timings for Magneticum Pathfinder

```
OpenMP active: max. 3 threads
MPI active with 8 tasks.

Total number of particles in file :
Type 0 (gas): 3428122059
Type 1 (dm): 3456649728
Type 2 (bndry): 0
Type 3 (bndry): 0
Type 4 (stars): 308093649
Type 5 (BHs): 495023

Total wall clock time for 'Splotch total time': 443.1826s
+- Input : 78.26% (346.8223s)
+- Rendering : 8.30% ( 36.7756s)
| +- Rendering proper : 81.88% ( 30.1129s)
| +- Chunk preparation : 17.64% ( 6.4857s)
| +- <unaccounted> : 0.48% ( 0.1770s)
+- Particle ranging : 7.02% ( 31.1301s)
+- 3D transform : 2.76% ( 12.2480s)
+- Particle coloring : 2.42% ( 10.7179s)
+- Output : 0.15% ( 0.6747s)
+- Post-processing : 0.08% ( 0.3399s)
+- Setup : 0.00% ( 0.0040s)
+- Particle sorting : 0.00% ( 0.0000s)
+- <unaccounted> : 1.01% ( 4.4701s)
```




z=6.6442

Universe 4D


300
280
260
240
220
200
y [Mpc]
220
240
260
280
300
z [Mpc]
300
280
260
240
220
200
x [Mpc]
220
240
260
280
300
z [Mpc]